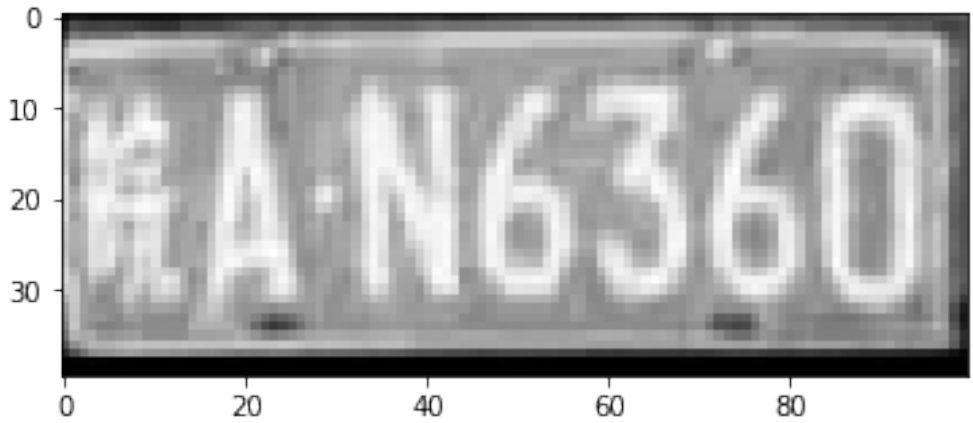
```
import json
api_token =
{"username":"iobananaoi","key":"0cbdda23beeb18cbb57e9e88bf26bfd1"}

with open('/root/.kaggle/kaggle.json', 'w') as file:
    json.dump(api_token, file)

!chmod 600 /root/.kaggle/kaggle.json

import cv2
import os
import matplotlib.pyplot as plt
import zipfile
import torch
from torch import nn
from torchvision.transforms import ToTensor, Compose, Normalize
from torch.utils.data import Dataset, DataLoader
import kaggle
from functools import reduce
import numpy as np
import PIL
from itertools import groupby

!kaggle datasets download -d romanleo2003/labtinkoff

Downloading labtinkoff.zip to /content
100% 2.12G/2.13G [01:35<00:00, 24.9MB/s]
100% 2.13G/2.13G [01:35<00:00, 23.8MB/s]

zip_ref = zipfile.ZipFile("/content/labtinkoff.zip", 'r')
zip_ref.extractall()

for name in zip_ref.namelist():
    old_name = "/content/" + name
    new_name = name.encode("cp437").decode('utf-8')

    zip_ref.extract(name)
    os.rename(name, new_name)

device = "cuda" if torch.cuda.is_available() else "cpu"
device

{"type":"string"}

batch_size = 64
```

Data processing

```
class CCPD(Dataset):
    def __init__(self, img_dir, transform=None):
        self.img_pathes = [f for f in os.listdir(img_dir) if
```

```
os.path.isfile(os.path.join(img_dir, f))][:64000]
        self.img_labels = [label.split("-")[1][:-4] for label in
self.img_pathes][:64000]
        self.img_dir = img_dir
        self.transform = transform

        self.unique_symbols = set([s for label in self.img_labels for
s in label])
        self.symb2idx = {list(self.unique_symbols)[i]: i+1 for i in
range(len(self.unique_symbols))}
        self.idx2symb = {i+1: list(self.unique_symbols)[i] for i in
range(len(self.unique_symbols))}

    def convert_label2idxs(self, label: str) -> list:
        return [self.symb2idx[symb] for symb in label]

    def convert_idxs2label(self, idxs: list) -> str:
        symbs_list = [self.idx2symb[idx.item()] for idx in idxs]
        return reduce(lambda x, y: x + y, symbs_list)

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_pathes[idx])
        image = cv2.imread(img_path)
        image = cv2.cvtColor(np.array(image), cv2.COLOR_BGR2GRAY)
        image = cv2.resize(image, (100, 40))
        label =
torch.tensor(self.convert_label2idxs(self.img_labels[idx]))

        if self.transform is not None:
            image = self.transform(image)

        return image, label

train_data = CCPD("CCPD2019-dl1/train", transform=ToTensor())
test_data = CCPD("CCPD2019-dl1/test", transform=ToTensor())

train_dataloader = DataLoader(train_data, batch_size=batch_size)
test_dataloader = DataLoader(test_data, batch_size=batch_size,
shuffle=True)

example_features, example_labels = next(iter(train_dataloader))
img = example_features[0].squeeze()
label = example_labels[0]
print(f"Features shape: {example_features.shape}")
print(train_data.convert_idxs2label(label))
plt.imshow(img, cmap="gray")
plt.show()
```

```
Features shape: torch.Size([64, 1, 40, 100])
皖AN6360
```



Model structure

```
class Encoder(nn.Module):
    def __init__(self, dropout):
        super(Encoder, self).__init__()

        self.layer1 = nn.Sequential(nn.Conv2d(1, 32, kernel_size=(3,
3), stride=1, padding=1),
            nn.PReLU(), nn.MaxPool2d(kernel_size=2, stride=1))
        self.layer2 = nn.Sequential(nn.Conv2d(32, 64, kernel_size=(3,
3), stride=1, padding=1),
            nn.PReLU(), nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer3 = nn.Sequential(nn.Conv2d(64, 128, kernel_size=(3,
3), stride=1, padding=1),
            nn.PReLU(), nn.MaxPool2d(kernel_size=2, stride=1))
        self.dropout = nn.Dropout(dropout)
        self.layer4 = nn.Sequential(nn.Conv2d(128, 128,
kernel_size=(3, 3), stride=1, padding=2),
            nn.PReLU(), nn.MaxPool2d(kernel_size=2, stride=1))

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.dropout(x)
        x = self.layer3(x)
        #x = self.layer4(x)

        x = x.reshape(x.shape[0], -1, x.shape[2]*x.shape[3])

        return x
```

```
class Decoder(nn.Module):
    def __init__(self, latent_dims, num_classes, num_layers, dropout):
        super(Decoder, self).__init__()

        self.lstm = nn.GRU(latent_dims, latent_dims, num_layers,
dropout=dropout)
        self.fc = nn.Linear(latent_dims, num_classes)
        self.sm = nn.Softmax(dim=1)

    def forward(self, x):
        out, state = self.lstm(x)
        logits = self.fc(out)
        preds = self.sm(logits)

        return preds

class CRNN(nn.Module):
    def __init__(self, latent_dims, batch_size, num_classes,
rnn_layers=3, rnn_dropout=0.01, cnn_dropout=0.01):
        super(CRNN, self).__init__()

        self.encoder = Encoder(cnn_dropout)
        self.decoder = Decoder(latent_dims, num_classes, rnn_layers,
rnn_dropout)

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

Training part

```
num_classes = len(train_data.unique_symbols) + 1 # Reserve one more
for blank symbol

def train(model, dataloader, optimizer):
    size = len(dataloader.dataset)
    model.train()

    for i, (img, label) in enumerate(dataloader):
        img, label = img.to(device), label.to(device)

        # Pred computation
        pred = model(img)
        pred = pred.permute(1, 0, 2)

        ctc_loss = nn.CTCLoss(reduction='mean', zero_infinity=True)
        input_lengths = torch.IntTensor(batch_size).fill_(23)
        target_lengths = torch.full(size=(batch_size,), fill_value=7,
```

```
dtype=torch.long)
        loss = ctc_loss(pred, label, input_lengths, target_lengths)

        #Backprop
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if i % 100 == 0:
            loss, current = loss.item(), i * len(img)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")

model = CRNN(931, batch_size, num_classes, rnn_layers=2,
rnn_dropout=0.25, cnn_dropout=0.25).to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

for i in range(10):
    print(f"========== Epoch {i+1}: ==========")
    train(model, train_dataloader, optimizer)
```

```
========== Epoch 1: ==========
loss: -2.670928  [    0/64000]
loss: -2.648199  [ 6400/64000]
loss: -2.665310  [12800/64000]
loss: -2.665378  [19200/64000]
loss: -2.661032  [25600/64000]
loss: -2.655488  [32000/64000]
loss: -2.654016  [38400/64000]
loss: -2.662504  [44800/64000]
loss: -2.652750  [51200/64000]
loss: -2.653880  [57600/64000]
========== Epoch 2: ==========
loss: -2.645248  [    0/64000]
loss: -2.648199  [ 6400/64000]
loss: -2.665310  [12800/64000]
loss: -2.665378  [19200/64000]
loss: -2.661032  [25600/64000]
loss: -2.655488  [32000/64000]
loss: -2.654016  [38400/64000]
loss: -2.662504  [44800/64000]
loss: -2.652750  [51200/64000]
loss: -2.653880  [57600/64000]
========== Epoch 3: ==========
loss: -2.645248  [    0/64000]
loss: -2.648199  [ 6400/64000]
loss: -2.665310  [12800/64000]
loss: -2.665378  [19200/64000]
loss: -2.661032  [25600/64000]
loss: -2.655488  [32000/64000]
```

```
loss: -2.654016  [38400/64000]
loss: -2.662504  [44800/64000]
loss: -2.652750  [51200/64000]
loss: -2.653880  [57600/64000]
```

Testing part

```
def test(model, dataloader):
    size = len(dataloader.dataset)
    model.eval()
    num_batches = batch_size

    test_loss, correct = 0, 0
    i = 0
    with torch.no_grad():
        for X, y in dataloader:
            if i == 156:
                break
            i += 1
            X, y = X.to(device), y.to(device)
            pred = model(X)
            pred = pred.permute(1, 0, 2)

            ctc_loss = nn.CTCLoss(zero_infinity=True)
            input_lengths = torch.IntTensor(batch_size).fill_(23)
            target_lengths = torch.full(size=(batch_size,),
fill_value=7, dtype=torch.long)
            test_loss += ctc_loss(pred, y, input_lengths,
target_lengths).item()

    test_loss /= num_batches
    print(f"Avg loss: {test_loss:>8f} \n")

test(model, test_dataloader)
```

```
Avg loss: -6.471286
```

На лицо очевидное недообучение модели. Скорее всего проблема в реализации выбранной архитектуры. Из идей, как можно было бы привести её к удобоваримому виду: 1) Заменить GRU, на LSTM(bidirectional=true), требует больше времени для обучения, но работает лучше 2) Добавить эпох 3) Как-то поменять decoder, добавить нормализацию по батчу, добавить больше конволюционных слоёв, увеличить окно ядра 4) Добавить слоёв в LSTM